# COSC 2231 - Programming 1
# Week 1 Laboratory Exercises

### Programming 1 Resources/Simple Java Programming

**Use JCreator to type/edit program source code**

1. Write a program which produces the following output on screen:

```
This is my first try
at programming in Java
It is fun!
```

   Use three different string literals to define the lines of the message.

2. Write a program which produces the following output onto the screen (note the blank lines):

```
This is my first try
at programming in Java
It is fun!
```

   You should again use three different string literals to define the lines of the message.

3. Additional Exercise

   (If you have already had a tutorial)

   If you have time - copy the code from question 7 of the week 1 tutorial and debug it - taking particular note of the compiler messages generated for the different errors in the code segment. (If you have not had a tutorial then it is suggested that you attempt this exercise in your own time after attending your tutorial).

   The relevant code segment is shown below:

```
//  /*  programs need
        comments  */
class OneOne
{
   public static void main(String[] args)
   {
      aString = new String("Well!");
      int  2ndString = "Now!";

      System.out.println("Isn't this an interesting program);
      System.out.println("Answer yes or no!")
   }
}
```

# COSC 2231- Programming 1
# Week 2/3 Laboratory Exercises

### Basic data types, operators and expressions

**You should work through the week one laboratory exercises and/or the eclipse tutorial if you have not already done so.**

**Once you have completed those tasks you can either practise writing programs in eclipse by coding up some of the examples you have seen in the tutorials and lectures or you can start working on the lab exercises below.**

**These exercises are similar in many respects to the task requuired for assignment 1 so they will be good practice for the task you will have to complete later on.**

1. **Write a program to read the current (I) in amperes and voltage (V) in volts and to compute the resistance (R) in ohms.**

   **Note: Resistance is given by R = V/I**

```
Enter V in volts: __

Enter I in amperes: __

Resistance R is: __ ohms
```

2. **Write a program that will read the marks of 3 students using the Scanner class and display the following:**
   - **(a) average marks**
   - **(b) maximum marks**

   **Hint: x = Math.max(a,b); will cause x to be assigned the maximum of a and b**
   **y = Math.max (x, c); will cause y to be assigned the maximum of a, b and c**

   **Find other such methods in Math class using Java docs (ask lab-assistant)**
   - **(c) minimum marks**

     **Similar to part b) but you need to use a different method**

3. **Write a program that prompts the user for the two (unequal) sides of a rectangle and then computes and prints the area and perimeter of the specified rectangle. The prorgram should display the output in the format below (in one line) using field width of 8 and precision of 2 digits after the decimal point using printf.**
   **Side1 = dd.dd Side2 = dd.dd Area = ddd.dd Perimeter = dddd.dd**

***Important: You should now use the quiz facility to review the first two chapters.***

# COSC1073/2362 – Programming 1
# Week 4 Laboratory Exercise

**While you are waiting to demonstrate your assignment 1 program you should work on the following lab exercise.**

1. **Write a program to convert car consumption from one unit to another. Offer the user the choice of either conversion, kpl to mpg or mpg to kpl. The interface should be:**

```
PREFERRED CONVERSION:

1. kpl to mpg
2. mpg to kpl

Enter your choice: 2

Enter miles per gallon value: mpg_value
The equivalent kilometres per litre is: kpl_value
```

**Use the constants 1 mile = 1609 metres and 1 gallon = 3.785 litres.**

**Useful formulae:**

```
mpg = (kpl / 1.609) * 3.785
kpl = (mpg * 1.609) / 3.785
```

**Note:** **the output of the program above should be formatted to 2 decimal places using the method** `System.out.printf()`**.**

# COSC1073/2362 – Programming 1
# Week 5 Laboratory Exercises

**Q1. Write an <u>iterative</u> program to accept an amount to invest, an annual interest rate and an investment period in years, and produce a table of values over the period. The program should iterate using a for loop and repeated multiplications (not powers) to do the calculations. The user interface for 10 years should be:**

```
Enter amount to invest:
Enter interest rate:
Enter number of years:

The value after 1 years is: ...
The value after 2 years is: ...
......
The value after 10 years is:...
```

**The formula for annual maturity investments with an interest rate of 5%**

**(Interest_Rate = 0.05) is:**

$$\text{Value} = \text{Initial\_Amount} \ (1 + \text{Interest\_Rate})^{\text{Years}}$$

**You can implement this calculation by using a loop which multiplies the investment value by a value of (1 + interest_rate) and resets the investment value to the result of the calculation each time.**

**This number of times this loop should repeat is dictated by the length of the investment period that was supplied.**

**Q2. An integer number N is prime if it is not divisible by any integer number K such that $K^2$ <= N. To test whether an integer N is prime your program should:**

- **if the number is –1, exit**
- **if the number is negative (but not –1), 0 or 1, display that and start over**
- **loop with the numbers 1....K while (K * K) <= N**
- **if any of the numbers divides N, then N is not prime; display that and start over**
  ***(NOTE: you can use the remainder operator "%" to determine if the factor you are examining divides into the number you checking for being prime).***
- **otherwise N is prime, display that**

**Write an <u>iterative</u> program to accept a positive integer as input and determine whether or not the integer is a prime number. The program should only accept positive integers, and should quit when a negative number is entered. An example of a user session could be:**

```
Enter an integer number: 61
61 is a prime number
Enter an integer number: -21
Wrong number, integer must be positive
Enter an integer number: 21
21 is not a prime number
Enter an integer number: -1
```

<u>Note</u>: there is no need to use any function of the mathematics package for these exercises.

# COSC1073/2362 – Programming 1
# Week 6 Laboratory Exercises

**Q1. Implement the Account class discussed in the lectures. Test the Account class using the driver class outlined below – you should copy and paste this code into a file called "TestAccount.java".**

**Once you have done this compile and run the TestAccount class.**

```
public class TestAccount
{
   public static void main(String args[])
   {
      Account mum = new Account("s123","Mercy Brown",1000.0);
      Account dad = new Account("g234","David Brown",2000.0);

      if (mum.withdraw(100) == false)
         System.out.println("mum unable to withdraw");

      dad.deposit(150);
      dad.transfer(mum,500);
      System.out.println("mum bal = "+mum.getBalance());
      System.out.println("dad bal = "+dad.getBalance());
   }
}
```

**Q2. In this exercise you should write another class similar to the Account class called SAccount to model a savings account.**

**The main difference between a "normal" account and a savings account is that there is a minimum amount that the balance cannot go below for a savings account.**

**Implement the SAccount class by doing the following:**

- **As well as defining instance variables for the account name and balance (as shown in the Account class), include an additional instance variable for the minimum amount.**

  *(examples of instance variable declarations can be found on page 213 of the course notes)*

- **Write a new constructor which, in addition to accepting a name and balance, also accepts a third argument, the minimum amount to be maintained.**

  *(an example of a constructor definition can be found on page 220 of the course notes)*

- **Include the accessors for the account name and balance as shown in the Account class.**

  *(examples of accessor definitions can be found on page 219 of the course notes)*

- **Define a new accessor for the minimum amount called "getMinAmount()".**

  *(Hint: refer to the accessor for the balance – "getBalance()" – the accessor for the minimum amount will be similar)*

- **Define the deposit method as shown in the Account class.**

- **Define the transfer method as shown in the Account class.**

  *(the definitions for the deposit and and transfer methods can be found on pages 216 and 218 of the course notes respectively)*

- **Implement the withdraw() method so that it ensures that the balance is always greater than the minimum amount that is stored in the corresponding instance variable.**

  *(Hint: the definition of the withdraw() method for the Account class on page 217 of the course notes checks to make sure that the balance is greater the withdrawal amount - you will need to something similar in your SAccount class which checks that the balance minus the minimum balance is greater than or equal to the withdrawal amount).*

- **Define a new method called addInterest() which accepts the interest rate as an argument, calculates the interest by multiplying the balance by the interest rate and adds the interest amount to the balance.**

  *NOTE: The interest rate will be supplied as a percentage out of 100, so you will need to divide the value that is passed in by 100 when doing your calculation.*

**Once you have implemented your SAccount class save it (to a file called "SAccount.java").**

**Copy and paste the following application class code into a file called "TestSAccount.java", save the file and then compile and run this class.**

**Notice that the withdraw() and transfer() methods should fail (return false) if withdrawing the specified amount will leave the balance less than the minimum amount to be maintained. Verify that final balances printed are the correct values.**

# COSC1073/2362 - Programming 1
# Week 7 Lab Exercises

**Test the SAccount class developed in the week 6 lab exercises by implementing an application that creates and uses SAccount objects as follows:**

**a) Create an array SAccount references that can hold up to five (5) SAccount objects.**

**b) Create 5 new SAccount objects based on the information shown below and store the objects in the array.**

| Account ID | Account name | Account balance | Minimum balance |
|---|---|---|---|
| S1234 | Dan | 1000 | 100 |
| S2435 | Marcia | 3000 | 750 |
| S4783 | Bobby | 6500 | 2000 |
| S1592 | Kim | 2200 | 600 |
| S7730 | Nancy | 4500 | 1200 |

**c) Deposit $100 to all of the SAccount objects that have a balance greater than $3000 (use decision statement nested inside a loop to find them)**

**d) Add 5% interest to all of the SAccount objects.**

**e) Display the id, name, and total available funds for each SAccount object in the array.**

**(The total available funds for the SAccount class is the balance minus the mimumum balance)**

# COSC 2231 - Programming 1
# Week 8/9 Lab Exercises

**While you are waiting to demonstrate your assignment 3 program you should start exploring the new concepts covered in assignment 4 by attempting the following lab exercise.**

1) **Implement the CAccount class discussed during the tutorial.**

2) **Write an application that addresses the following requirements:**

   **(make sure you have a copy of the Account class and the SAccount subclass that were discusssed previously in weekk 5 and 7 in the lectures)**

   *NOTE: Points that are labelled "Week 8" deal with concepts covered in the week 7 lecture and points labelled "Week 9" deal mainly with polymorphism and object typecasting (covered in the week 8 lecture).*

   a) **(Week 8) Create an array of Account references that can hold up to six (6) Account (or Account subclass objects).**

   b) **(Week 8) Fill the array with the following objects:**

| Account Type | Account ID | Account name | Account balance | Minimum balance | Overdraft limit |
|---|---|---|---|---|---|
| Account | A1234 | Bob | 1000 | | |
| SAccount | S2435 | Jill | 3000 | 750 | |
| CAccount | C6451 | Jim | 6500 | | 2000 |
| SAccount | S1624 | Sally | 2800 | 1200 | |
| CAccount | C1198 | Joan | 4500 | | 2500 |
| Account | A4467 | Mike | 2500 | | |

   c) **(Week 8) Print the basic details details (id, name, balance) for each object in the array to the screen.**

   d) **(Week 8) Deposit $100 to all accounts that have a balance greater than $3000**

   e) **(Week 8) Transfer $200 from Bob to every other account.**

   f) **(Week 9) Add 2% interest to all SAccount objects - use a loop to locate them**

   g) **(Week 9) Deduct charges from all CAccount objects - again use a loop to locate them.**

   h) **(Week 9) Display the Account type, id, name, and total available funds for each object in the array.**

      **The total available funds for each of the accounts can be calculated as follows:**

      - **For the Account class it is the balance amount**
      - **For the SAccount class it is the balance minus the mimumum balance**
      - **For the CAccount class (discussed in the tutorial) there is a method you can call which returns the total available funds**

**Once you have completed this exercise you should start working on assignment 4, which is**

**similar in many ways in terms of general structure and the concepts involved.**